Electronic Version
Stylesheet Version v1.1.1

# Description

# METHOD AND SYSTEM FOR DETERMINING ENGINE STATE OF A HYBRID ELECTRIC VEHICLE

## BACKGROUND OF INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates to hybrid electric vehicles, and more particularly to a method and system for managing the various inputs that determine the on/off state of the engine in a hybrid electric vehicle.

[0003] 2. Background Art

[0004] A typical powertrain for a hybrid electric vehicle has at least two sources of power. One source typically is an internal combustion engine, and the second source is a high voltage electric motor, usually an induction motor. There are three general classes of hybrid electric vehicle powertrains; i.e., parallel, series and parallel-series.

[0005] A parallel hybrid electric vehicle powertrain usually in-

cludes an internal combustion engine, one or more electric motors, and an energy storage device, usually a battery. Mechanical power from both the engine and the electric motor(s) is directly available to propel the vehicle.

[0006] A series hybrid electric powertrain includes an engine, a generator, an electric motor, and an energy storage device, usually a battery. The electric motor is the only source of mechanical power that is directly available to propel the vehicle. Power from the engine is electrically routed from the generator to the electric motor.

[0007] A parallel-series hybrid electric powertrain configuration is a combination of parallel and series configurations. It is capable of operating in a first operating mode with characteristics of a parallel hybrid electric powertrain and in a second operating mode with characteristics of a series hybrid electric powertrain.

[0008] Each of the three powertrain configurations described previously is capable of propelling the vehicle by power from the electric motor power alone, with or without the engine running to produce power. Typically there are many factors that determine whether it is desirable to run the engine or not, including those affecting safety, fuel economy, performance, and passenger comfort. An elec-

tronic vehicle controller with powertrain control software is responsible for considering these factors and making the complex decision to run the engine or not.

[0009] Because of the large number of inputs that must be evaluated by the powertrain control module as factors in determining the desired engine state, the control module software architecture must provide a means of organization and prioritization of these inputs. In addition, due to the fact that the inputs may change throughout the design phases of vehicle development, the software architecture must be flexible enough to maintain this organization in spite of modifications.

## SUMMARY OF INVENTION

[0010] In accordance with the present invention a method and system of engine state determination is provided that is based on inputs from various vehicle sources. Each of a plurality of requestors compares vehicle or environmental conditions to a requirement and outputs a request state variable with a fundamental request state. The requestors are grouped according to function or component as appropriate to the implementation. The request state variables are combined and simplified at the output of each group. A state machine receives the final combined and

simplified request state, and evaluates it according to the active or inactive status of the requests. The state machine carries out the proper state transition and outputs the desired engine state. Conventional engine control algorithms receive the desired engine state and start or stop the engine as necessary.

[0011] The present invention offers an organized, efficient, and effective method of managing engine state in a hybrid electric vehicle. The invention is capable of handling an unlimited number of requirements on engine state and accommodates changing requirements with minimal impact on the software architecture and organization. The invention provides clear traceability between vehicle requirements and software implementation, each requirement being handled by a single requestor that generates a specific fundamental request state. The requests may be grouped into hierarchical levels through combination and simplification operations allowing requirements such as safety, driveability, fuel economy, and other customer expectations to be associated and organized in a manner that is appropriate to the design of the vehicle. In all cases, including those where the requests conflict, the engine state is entirely predictable and stable. Requests may

be prioritized so that the most important requirements such as those for safety always take precedence over less important requirements.

## Brief Description of Drawings

[0012] Figure 1 is a schematic representation of a parallel-series powertrain system configuration;

[0013] Figure 2 is a schematic block diagram of one embodiment of the invention showing the request flow and arbitration;

[0014] Figure 3 is a table of requests and desired state machine transitions as a function of engine state;

[0015] Figure 4 is a diagram showing the request prioritization employed in the present invention;

[0016] Figure 5 is a logic diagram showing the process of joining two or more request states together into one request;

[0017] Figure 6 is a flow chart depicting simplification of many request states into fundamental request states; and

[0018] Figure 7 is a state diagram showing the on/off state of the engine and the logic expressions that determine engine state.

## Detailed Description

[0019] Referring now to the drawings and initially to Figure 1, a typical hybrid electric vehicle powertrain of the parallel-se-

ries configuration is shown. One example of this configuration is shown and described in United States Patent Application S.N. 10/248,886, filed February 27, 2003, which is assigned to the assignee of the present invention and incorporated herein by reference. The invention described or claimed herein, however, is not limited to the parallel-series configuration of Figure 1. The configuration includes a powertrain control module 10 that communicates electronically with a battery 12 and an engine 14, as well as a motor 16 and a generator 18, within a transmission 20. The transmission includes a planetary gear unit 22 that distributes torque through step ratio gears 24 to an output shaft 26 that drives vehicle traction wheels 28 through a differential and axle mechanism 29. The step ratio gears are mounted on a countershaft, with one of the gears engaging a gear that is driven by the motor 16, which acts as a torque input for the countershaft gearing.

[0020] The powertrain control module 10 is responsible for managing power flow, operating the engine 14 and other components, and maintaining the state of the vehicle including thermal conditions, faults and failures, and the state of charge of battery 12. The powertrain control module 10 includes a microprocessor and supporting

electronics to run a software program. The software program includes a plurality of algorithms or modules and state machines for carrying out the aforementioned tasks.

[0021] The following terms are defined for use in describing the present invention.

[0022] ·engine state – a value which describes the engine condition, either on or off. Engine state is represented herein by the state variable eng_st. A value of one corresponds to engine on and zero corresponds to engine off.

[0023] ·request – an expressed desire for a particular action with respect to engine state. There are six requests, FPD, FPU, IPU, PU, IPD, and PD. These are depicted in Figure 3. Each request has a particular meaning in each engine state.

[0024] ·request variable – a bit quantity which indicates the active or inactive status of a request. The request variables are labeled fpd, fpu, ipu, pu, ipd, and pd, corresponding to the six requests. A value of one indicates that the corresponding request is active and zero indicates that the request is inactive.

[0025] ·request state – a value which contains a field of six bits indicating the active or inactive status of each of the six engine requests.

[0026] ·fundamental request state – one of the nine request

states which are not redundant and do not conflict.

[0027] ·request state variable – a quantity which represents a request state. Request state variables are named in the form xxx_req_st.

[0028] ·requirement – a need for a particular fundamental request state to be generated. For example: "A request state of PULL_DOWN_REQ shall be generated if the battery state of charge is greater than 85 percent."

[0029] ·requestor – a software construct which compares vehicle or environmental conditions to requirements. A requestor outputs a request state variable with the appropriate fundamental request state.

[0030] ·combination – the process of joining two or more request states into one request state.

[0031] ·simplification – the process of reducing a request state into a fundamental request state.

[0032] arbitration – the process of following requests according to priority.

[0033] The fundamental request states are listed below in Table 1.

Table 1: Definition of fundamental request states

| Request State | fpd | fpu | ipu | pu | ipd | Pd |
|---|---|---|---|---|---|---|
| FORCE_PU | 1 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LL_DOWN_REQ | | | | | | |
| FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |
| IN-HI-BIT_PULL_UP_REQ | 0 | 0 | 1 | 0 | 0 | 0 |
| PULL_UP_REQ | 0 | 0 | 0 | 1 | 0 | 0 |
| IN-HI-BIT_PULL_DOWN_REQ | 0 | 0 | 0 | 0 | 1 | 0 |
| PULL_DOWN_REQ | 0 | 0 | 0 | 0 | 0 | 1 |
| NO_REQ | 0 | 0 | 0 | 0 | 0 | 0 |
| IN-HI-BIT_CHANGE_REQ | 0 | 0 | 1 | 0 | 1 | 0 |
| PULL_DWN_INH_PULL_UP_REQ | 0 | 0 | 1 | 0 | 0 | 1 |

[0034] Each fundamental request state comprises a 6 bit binary word that identifies a request state by setting the requests variables fpd, fpu, ipu, pu, ipd or pd to a 1 or a 0, which indicates whether a request is active or inactive. Each of the six requests, FPD, FPU, IPU, PU, IPD, and PD has a particular meaning in each engine state as depicted in Figure

3. The requests identified by uppercase, label the types of requests, as in "do this ... ". The request identified by lowercase, are the request variables that indicate whether a request is active or inactive. Thus, if fpu = 0, then the FPU request as defined in Figure 3 is inactive. These requests are defined with a fixed priority as shown in Figure 4. In a case where the requests conflict, arbitration is carried out as follows:

[0035] FPD (Force Pull Down) is followed if there is an FPD request (the state of FPU, IPU, PU, IPD, and PD does not matter)

[0036] FPU (Force Pull Up) is followed if there is an FPU request and there is no FPD request(the state of IPU, PU, IPD, and PD does not matter)

[0037] IPU (Inhibit Pull Up) is followed if there is an IPU request and there is no FPU request(the state of FPD, PU, IPD, and PD does not matter)

[0038] PU (Pull Up) is followed if there is a PU request and there is no FPD request or IPU request (the state of FPU, IPD, and PD does not matter)

[0039] IPD (Inhibit Pull Down) is followed if there is an IPD request and there is no FPD request(the state of FPU, IPU, PU, and PD does not matter)

[0040] PD (Pull Down) is followed if there is a PD request and there is no FPU request, IPD request, or PU request (the state of FPD and IPU does not matter)

[0041] Referring now to Figure 2, one embodiment of the invention is shown to comprise a plurality of requestors or software constructs, each of which compares vehicle or environmental conditions to a requirement and outputs a request state variable with a fundamental request state. The requestors are grouped according to function or component as appropriate to the implementation. The request state variables are combined and simplified at the output of each group. The requestors and combination and simplification blocks are implemented as pieces of software code that have inputs and outputs and perform the described functions.

[0042] In the embodiment of the invention shown in Figure 2, there are, by way of example, four requestors 30-36 each with a specific requirement. Requestor 30 generates a request state variable of keyoff_req_st, which corresponds to the fundamental request state FORCE_PULL_DOWN_REQ if the ignition key is in the OFF position, and corresponds to the fundamental request state NO_REQ if the ignition key is in the RUN or START position. Requestor 32 gener-

ates a request state variable of keyon_req_st, which corresponds to the fundamental request state of FORCE_PULL_UP_REQ for 10 seconds after the ignition key is first turned from the RUN to the START position during the current driving cycle and otherwise corresponds to the request of NO_REQ. Requestor 34 generates a request state variable of gear_req_st, which corresponds to the fundamental request state of INHIBIT_CHANGE_REQ for 2 seconds after a gear selector change is detected and otherwise corresponds to the request of NO_REQ. Requestor 36 generates a request state variable of batt_req_st, which corresponds to the fundamental request state of PULL_DOWN_REQ if the battery state of charge is greater than 85 percent and corresponds to the request of NO_REQ if the battery state of charge is less than or equal to 85 percent.

[0043] In this embodiment of the invention, there are three groups. The first group 38, identified as Key Read, includes Requestor 30 and Requestor 32. The request state variables keyoff_req_st and keyon_req_st are combined at combination block 40 and simplified at simplification block 42 to create key_req_st. The second group 44, identified as Driver Input, includes the Key Read group 38

and Requestor 34. It combines request state variables key_req_st and gear_req_st at combination block 46 and simplifies the request state variables at simplification block 48 to create driver_req_st. The third and final group joins the Driver Input group 44 and the Battery Management group 50 by combining and simplifying driver_req_st and batt_req_st in the combination block 52 and the simplification block 54. The combination function performed at blocks 40, 46, and 52 is a process of joining two or more request states into a single request state. The simplification function performed at blocks 42, 48, and 54 is a process of reducing a request state to a fundamental request state.

[0044] A state machine generally indicated at 56 receives the final combined and simplified request state, final_req_st, and evaluates it according to the active or inactive status of the six requests. The state machine carries out the proper state transition and outputs the engine state. Engine state is a value that describes the engine condition, either on or off. Engine state is represented here by the state variable eng_st. A value of one corresponds to engine on and zero corresponds to engine off. Other engine control algorithms included in the control module 10 re-

ceive the engine state, eng_st and start or stop the engine as necessary.

[0045] A request state combination operation performed in the blocks 40, 46, and 52 is achieved with a bitwise OR operation on two or more request state variables. The operation is depicted generally in Figure 5 which shows the combination of a_req_st and b_req_st to produce c_req_st by ORing the respective single bit request states fpd, fpu, ipu, pu, ipd, and pd from a_req_st and b_req_st with OR gates 60-70 to produce c_req_st. In the combination operation, there is no loss of information with regards to requests. All active requests are carried through to the output.

[0046] There are 64 request states ($2^6$), but only nine of these states are fundamental request states. Many of these states are redundant or contain conflicting requests. For instance, a request state containing active requests for both FPD and PD is redundant because these requests express the same desire with respect to engine state (only at different priorities). Such a request state can be simplified to contain only an active request for FPD. Another example is a request state that contains active requests for both FPD and FPU, which conflict. Such a request state can

be simplified to contain only an active request for FPD, which has a higher priority than FPU. The state machine 56 is capable of evaluating any of the 64 request states. Simplification, whether performed prior to the state machine as shown or within the state machine, is useful because it aids the testing and debugging steps of vehicle controls development. Simplification reduces the number of requests that programmers and engineers must learn and interpret from 64 to 9.

[0047] Simplification is performed according to the flow chart in Figure 6. The process is designed such that there is no loss of information with regards to the meaning of a request state, only elimination of redundancy and conflicts. The simplifying operation selectively passes active requests according to a fixed priority and alters the inactive or active status of requests such that a minimal vocabulary of fundamental request states is sufficient to convey all request information. The process inputs a request state to a system of decision blocks 72-88. Each decision block evaluates one bit of the request state. A logic 1 value in decision blocks 72, 74, 78, 80, and 82 directly produce fundamental request states of FORCE_PULL_DOWN_REQ, FORCE_PULL_UP_REQ, PULL_UP_REQ, IN-

HIBIT_PULL_DOWN_REQ, and PULL_DOWN_REQ, respectively, at the simplification output. A logic value of 0 in all bits results in an output of NO_REQ. A logic value of 1 in decision block 76 results in further evaluation in blocks 84-88, leading to an output of INHIBIT_CHANGE_REQ, PULL_DOWN_INH_PULL_UP_REQ, or INHIBIT_PULL_UP_REQ.

[0048] While each individual requestor will output one of the fundamental request states the result of a combination operation on two or more requestor outputs (fundamental request states) may or may not be a fundamental request state. The simplification operation ensures that the two or more requestor outputs that were combined become a fundamental request state, not just any request state. Thus, the simplification operation makes the meaning of a request state more apparent after a combination operation.

[0049] The state machine 56 is implemented according to the state diagram in Figure 7. The state transition conditions follow from the meaning of requests specified in Figure 3 and the prioritization of requests depicted in Figure 4 and described previously. An engine state transition from 1 to 0 occurs if a FPD request is active or a PD request is active and the FPU, IPD, and PU requests are inactive. The active

or inactive state of IPU does not matter since the IPU request is meaningless in the engine on state, as shown in Figure 3. An engine state transition from 0 to 1 occurs only if the FPD request is inactive since FPD has higher priority than all other requests with which it conflicts. If FPD is inactive, then a transition from 0 to 1 will occur if the FPU request is active or the PU request is active and the IPU request is inactive. The actie states of IPD and PD do not matter with regard to the engine state transition of 0 to 1. The IPD request is meaningless in the engine off state, as shown in Figure 3. The PD request has a lower priority than all requests with which it conflicts, so the state of PD does not matter in the 0 to 1 transition.

[0050] With reference to Figure 2, the following explanation of the operation of the invention is given by way of example. Assume the engine is off; the driver has just switched the key position from RUN to START for the first time and has also shifted the gear selector. The battery state of charge is 90%. Based on the requirements listed for each of the requestor 30-36 and the vehicle conditions, there are conflicting requests. The requestors will output the following request states:

[0051] Requestor 30: keyoff_req_st = NO_REQ

[0052] Requestor 32: keyon_req_st = FORCE_PULL_UP_REQ

[0053] Requestor 34: gear_req_st = INHIBIT_CHANGE_REQ

[0054] Requestor 36: batt_req_st = PULL_DOWN_REQ

[0055] The Key Read group 38 will combine and simplify the values of keyoff_req_st and keyon_req_st as follows:

| Request State Variable | Request State | fpd | fpu | ipu | pu | ipd | pd |
|---|---|---|---|---|---|---|---|
| keyoff_req_st | NO_REQ | 0 | 0 | 0 | 0 | 0 | 0 |
| key_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |
| block 40 output | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |
| key_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |

[0056] The Driver Input group 44 will combine and simplify the values of key_req_st and gear_req_st as follows:

| Request State Variable | Request State | fpd | fpu | ipu | pu | ipd | pd |
|---|---|---|---|---|---|---|---|
| key_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |
| gear_req_st | INHIBIT_CHA | 0 | 0 | 1 | 0 | 1 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | NGE_REQ | | | | | | |
| block 46 output | Not a fundamental request State | 0 | 1 | 1 | 0 | 1 | 0 |
| driver_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |

[0057] The final stage will combine and simplify the values of driver_req_st and batt_req_st from the Battery Management group 50 as follows:

| Request State Variable | Request State | fpd | fpu | ipu | pu | ipd | pd |
|---|---|---|---|---|---|---|---|
| driver_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |
| batt_req_st | PULL_DOWN_REQ | 0 | 0 | 0 | 0 | 0 | 1 |
| block 52 output | Not a fundamental request state | 0 | 1 | 0 | 0 | 0 | 1 |
| final_req_st | FORCE_PULL_UP_REQ | 0 | 1 | 0 | 0 | 0 | 0 |

[0058] The state machine 56 reads the value of final_req_st and finds that fpd = 0, fpu = 1, ipu = 0, pu = 0, ipd = 0, and pd = 0. It makes the state transition from eng_st = 0 to eng_st = 1 according to the state diagram shown in Figure

7. It will be appreciated that in Figure 7 the notation "+" represents a logical OR operation, the "dot" represents a logical AND operation, and the "bar" represents a logical NOT operation. Thus, in order to transition from eng_st = 0 to eng_st = 1, it is necessary that either fpu OR pu equal 1 AND that ipu equals 0 AND further that fpd equals 0. The eng_st variable is outputted to other control algorithms that start the engine.

[0059] The invention is applicable not only to series, parallel and parallel/series hybrid electric vehicles but to any controlled device that requires an on/off decision with many inputs. This may include HVAC systems, manufacturing equipment, and robotics.

[0060] While the best mode for carrying out the invention has been described in detail, those familiar with the art to which this invention relates will recognize various alternative designs and embodiments for practicing the invention as defined by the following claims.